Pakistan Journal of Applied Sciences 1 (4) 446-451, 2001

Content Based Compression of Turkish Documents

Banu DIRI
Yildiz Technical University, Department of Computer Engineering
Yildiz 80750, Istanbul, Turkey

Abstract. The main goal of this study is to analyse the morphological structure of the Turkish documents. The new proposed method consists of lossless compressing the monograms, digrams, trigrams, roots-stems and suffixes individually using a statistical approach. In this work 1-gram, 2-gram, 3-gram, root-stem and suffix frequencies have been computed for Turkish language. A tuned template has been prepared for each group. The compression of Turkish documents has been performed by using the static Huffman Coding method and a compression performance of 39 % for 1-gram, 47 % for 2-gram, 51 % for 3-gram and 59 % for Word Based Dynamic Huffman has been measured.

Keywords: Text compression, Language modelling, Huffman coding, n-gram models, Turkish Corpus

Introduction

Data compression is simply the efficient digital representation of a source. Infact, data compression is the representation of a source in digital form with as few bits as possible while maintaining an acceptable loss in fidelity. The source can be data, still images, speech, audio, video, or whatever signal needs to be stored or transmitted. There are many known methods for data compression. They are based on different ideas and produce different results, but they are all based on the same principle, they compress data by removing redundancy from the original data in the source file. Data compression methods can be divided into two groups: *lossy* and *lossless* compression , Salomon, (1998). Lossless means perfect reconstruction of the source and lossy means that the source is not perfectly preserved in the representation.

Information Theory Concepts: Entropy: Entropy (H) and Redundancy (R) have important application areas such as compression, language identification, etc. The entropy of a single symbol a_i is defined as $-P_i\log_2 P_i$ Salomon, (1998) and Gibson *et al*, (1998) where P_i is the probability of occurrence of a_i in the data. The entropy of a_i is the smallest number of bits needed, on the average, to represent symbol a_i (eq.1).

$$-\sum_{1}^{n} P_i \log_2 P_i \qquad (1)$$

This is the smallest number of bits needed, on the average, to represent the symbol. The entropy of the data depends on the individual probabilities P_i, and is smallest when all n probabilities are equal. This fact is used to define the redundancy R in the data. It is defined as the difference between the entropy and the smallest entropy (eq.2).

$$R = \left(-\sum_{1}^{n} P_i \log_2 P_i\right) - \log_2\left(1/P\right) = -\sum_{1}^{n} P_i \log_2 P_i + \log_2 n \qquad (2)$$

One several quantities that are commonly used to express the efficiency of a compression method. It is the compression ratio (eq.3).

$$\text{Compression ratio} = \frac{\text{Size of the output stream}}{\text{Size of the input stream}} \qquad (3)$$

Sometimes the quantity $\eta=100*(1 - \text{compression ratio})$ is used to express the quality of compression.

Huffman Coding: This is commonly used method for data compression. It serves as the basis for several popular programs used on personal computers. Some of them use just the Huffman method while others use it as one step in a multi-step compression process.

The method starts by building a list of all the alphabet symbols in descending order of their probabilities. It then constructs a tree, with a symbol at every leaf, from the bottom up. This is done in steps where, at each step, the two symbols with smallest probabilities are selected, added to the top of the partial tree, deleted from the list, and replaced with an auxiliary symbol representing both of them. When the list is reduced to just one auxiliary symbol the tree is complete. The tree is then traversed to determine the codes of the symbols. The size of the Huffman code for symbol x is less than or equal $\lceil -\log_2 P_i \rceil$ Salomon (1998).

Huffman Decoding: The decoder must know what is at the start of the stream, by accessing the start of the stream it constructs the Huffman tree for the alphabet. After having constructing the Huffman tree the rest of the stream can be decoded. The algorithm for decoding is simple. Start at the root and read the first bit of the compressed stream. If it is zero, follow the bottom edge if it is one, follow the top edge. Read the next bit and move another edge toward the leaves of the tree. When the decoder gets to a leaf, it finds the original, uncompressed code of the symbol, and that code is emitted by the decoder. The process starts again at the root with the next bit.

Lossless Compression of Documents in Turkish: Firstly, a Turkish document has been separated into 1-grams (monograms), 2-grams (digrams - 2 letter combinations), 3-grams (trigrams - 3 letter combinations) and then has been compressed with the improved lossless compression method. Secondly, the words in a document in Turkish have been separated into root-stem and suffixes and then have been compressed with the improved lossless compression method. According to the Galgary and Canterbury corpus used in

the testing of the improved compression methods, the test-group formed by 14 different documents in Turkish has been used (Table1).

Table 1: The documents that forms the test-group

File Type	Raw Size (byte)	No	File Type	Raw Size (byte)
article1	24.623	8	novel2	236.778
article2	36.596	9	novel3	251.606
article3	41.378	10	article5	666.120
article4	48.880	11	article6	731.929
text1	93.884	12	article7	1.055.244
thesis	135.360	13	article8	1.795.215
novel1	160.239	14	article9	2.147.095

By Using Monograms, Digrams and Trigrams, lossless Compression of Documents in Turkish: In the first part, the frequencies of all the characters (1-gram) that can be found in a Turkish document have been computed. In this work, lower case and upper case letters have been considered individually. Lower case 29 letter, upper case 29 letters plus non Turkish letters (x, w, q) 3*2=6, together with the space and the special characters, a set of 89 characters have been taken in consideration. Frequencies, probabilities and the codelengths (codelen) of the 89 characters are shown in Table 2.

Each 1-gram character represented by a Huffman code. The entropy of the system is H=-4,37 bpc and the redundancy R=0,026. All of these data leads us to measure a compression performance of %39 when we implement all of the files in the test-group. Experimental results have been given in the Table 5.

The second part of the work, 2-gram processing has been considered. The 2-gram histogram of each file in the test-group has been collected and by filtering the most least used digrams a modified template has been taken in consideration. By merging the digram and the monogram template a new template has been set up. The new template contains 533 elements. The 1-gram part of the new template is for coding the digram character set which are not in the filtered digram template as 1-gram characters. The frequencies, probabilities and codelengths of the template are shown in the Table 3.

The most common digrams in Turkish documents are (ar, la, er, an, le, in, de, ýn, da, en, ...). Each digram character set and also the monogram character represented by a Huffman code. The entropy of the system is H=-6,63 bpc and the redundancy is R=0,033. The measurement of the compression shows us a %47 compression performance, implementing all of the files in the test-group. Experimental results have been given in the Table 5.

The third part of the work is for trigram compression method. The trigram character set histogram of each file in the test-group has been collected and by filtering the least used trigrams a modified template has been taken in consideration. By merging the trigram and the monogram template a new template with 602 elements has been set up. The frequencies, probabilities and the codelengths of the template are shown in the Table 4.

The most common trigrams in Turkish documents are (lar, ler, bir, eri, arý, yor, ara, nda, ini, ile,...). Each trigram character set and also the monogram characters represented by a Huffman code. The entropy of the system is H=-6,056 bpc and the redundancy is R=0,034. Implementing the trigram compression system on all of files in the test group leads us to measure a %51 compression performance. Experimental results have been given in the Table 5.

Comparison of Experimental Results (I. Step): The Huffman coding Method has been considered for compressing the text files Knut, (1985). 3 different algorithms have been developed for the compression. The analysis of the text has been done individually for 1-gram, 2-gram and 3-gram characters by using the appropriate compression algorithm. The compression performance (η) of the files in the test-group has been given in the Table 5. The (Fig.1) is the graphical layout. The average compression performance for monogram is %39, for digram is %47 and for trigram is %51. The compression performance varies depending on the length and the text data (for example the space character usage).

To improve the compression performance:

1. Instead of lower case and upper case characters, considering the text as a single case, would give a better result because of the ability of using a shorter template. But this is not a real compression implementation.
2. Considering the carriage return (#13) and the line feed (#10) characters, as a single character would give a better result.
3. Instead of symbolising individually lower case and upper case characters having two new characters in the template to differentiate the lower cases and upper cases would give a better result.

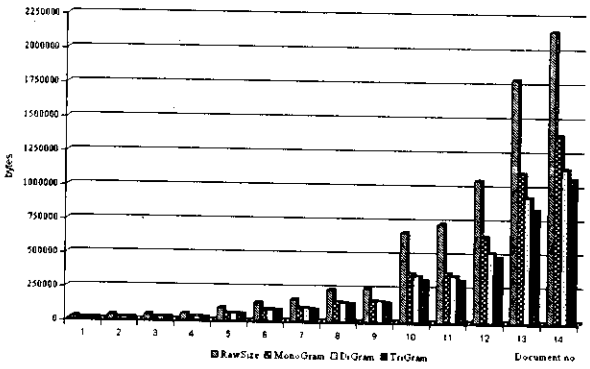

Fig 1: The graphics of the compression performances received from the three different methods

Compression of Words in a Document Written in Turkish: To increase the success of the compression method that we are trying to develop the morphological structure of Turkish and in order to code more data at once, the word separated its root-stem and suffixes. The derivative formed by adding forming suffixes to the noun or verb roots is called the *stem* (like aç-lýk, bak-ý-m, gör-ü-þ). It is possible to code more data at once by analysing the words divided into the stem and suffixes. In a word there may be the root and the stem at the same time, in this case the stems which are longer than the roots are taken in consideration. For example, let's say that both "ay" and "aydýn" are included in the

dictionary. If the word which will be separated is "aydýnlýk", we should take "aydýn" instead of "ay" as the root. That means "aydýnlýk" should be written as aydýn:root and lýk:suffix. The method doesn't need a data, which uses the knowledge on frequency of use of the root-stem, and suffixes are taken from the document which is going to be compressed if only they still stay related to the file. So the Huffman code word for the same word that will be the most suitable for every document's characteristic is found Diri, (1999). The method developed for that is called the dynamic Huffman algorithm.

Word Based Dynamic Huffman Method: We need two different dictionaries of root-stem and suffixes in order to separate the word into root-stem and suffixes. In the formation of root-stem dictionary, a root dictionary that includes 13.206 roots and stems by using Turkish dictionary Kurumu, (1992) and thesaurus Kurumu (1996). A dictionary of suffixes has been formed which includes all the suffixes (522) and all the control data to be used in the separation of lower case and upper case letters. All the information in root and suffix dictionaries is stored with lower case and the analysis of the document is done and coded according to the lower case. In the decompression of a file, in order not to lose its original form, we have to know which letters are used in the suffix dictionary. If all the letters are upper case or there is one or more upper case letters in the word, these control characters are used to find their places in the word.

After the prepared root-stem and suffix dictionaries are stored in the memory, the document that will be compressed is put into the memory in blocks and every character in each block examined. At the end of each block, the next block is read automatically in the memory and when a word starts in a block and ends in the next that means it is faultless. A modular finite state machine has been developed in order to accelerate the examining of characters in blocks. All the phases starting from the output of two machines to the input of the Huffman trees is shown in the block diagram in Fig. 2.

When the finite state machine finds a word in a block, it separates the word into the root-stem and suffixes in the most appropriate way by calling the most necessary algorithms. As the words are analysed by changing into lower case letters for the lossless compression of the upper case letters, all the necessary control data has to be formed. And when a word in Turkish or any other language that is not in the root-stem dictionary used during the analysis, this word is coded character by character. Because of this, during the compression, the developed system adjusts to special occasions. As it can be seen Fig. 2, two Huffman trees are formed. One form Huffman codes for the root-stem ® Huffman tree) and the other for suffixes (S Huffman tree). In both trees, as the code confliction is possible, in order to prevent the confusions during the decoding "0" bit before the Huffman codes in "R" tree, and "1" bit before the Huffman codes in "S" tree is added.

In the decompression of a coded compressed data, "R" and "S" trees are used. The first bit of the coded bit files shows which Huffman tree the code should be taken from. If it is "0", the "R" tree and if it is "1", "S" tree should be used in the process. After this bit control, the shortest code length of the tree is taken into consideration and is searched on the bit file. If the match between the root-stem or suffix which has this code value, the following bit used to decide which of "R" or "S" trees will be used in the process. If the match can't be made, the process continues by increasing the length of code searched till the match can be made. For the lossless decompression of the data compressed with codes, the decoder needs the header data, which is added to the compressed data. The format of header data can be seen in Fig. 3.

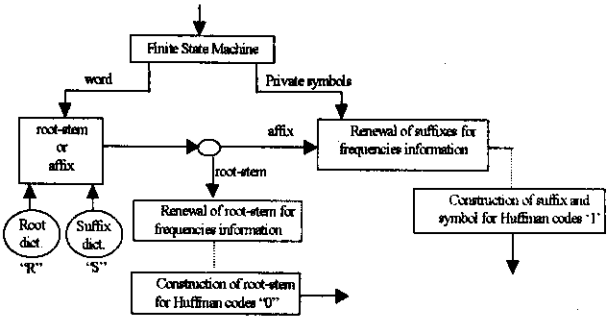

Fig. 2: The block diagram of the system used to analyse the document and form the code words

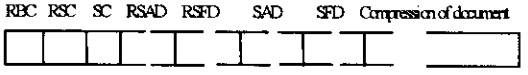

Fig. 3: Format of the header

RBC (residue-bit-count) has a 1 byte length and when the last value is less than 8 bits; it shows how many bits of the value in last byte to be decoded as real data. The root-stem number received after the analysis of the document is stored in RSC and the number of suffixes is stored in SC. The field in which the root-stem address data is stored, RSAD, shows the dictionary address of every root-stem that can be found in the documentary and dictionary. As there are 13.206 words in the root-stem dictionary used, address data of each is stated with $\lceil \log_2 13.206 \rceil$ bits. The total length of this field in the header is $\lceil 14*(\text{the number of root-stem in the document})/8 \rceil$ bytes. Suffix-address data (SAD) includes the address data of the suffixes. As we have a total sum of 580 suffix data, each suffix can be expressed with $\lceil \log_2 580 \rceil$ bits and the field of suffix address information is $\lceil 10*(\text{the number suffixes in the document})/8 \rceil$ bytes long. In RSFD field, the information on Huffman tree that is formed to code the root-stem information is transferred and in SFD, the information on Huffman tree that is formed to code the suffix information is transferred.

Sending the tree code data: By transferring the data on Huffman tree codes instead of the frequency data of root-stem and suffixes which are at the beginning of the compressed file and in the header data, the following are done: (1). The average length of the header data is

Table 2 Frequencies, probabilities and codelengths of monograms

1 gram	Freq.	Codelen	P_i	1 gram	Freq.	Codelen	P_i
blank	2057045	2	0,277502	,	55048	7	0,007426
a	553340	4	0,074647	ç	50422	7	0,006802
e	433174	4	0,058437	ğ	49160	7	0,006632
i	407126	4	0,054923	h	44463	7	0,005998
n	350248	4	0,04725	v	44419	7	0,005992
r	339737	4	0,045832	.....	.....	.....	.....
l	310492	5	0,041886	O	5269	10	0,000711
ý	241875	5	0,03263	G	5257	10	0,000709
k	219736	5	0,029643	N	4553	11	0,000614
.....	.....	.....	.....	P	4432	11	0,000598
z	71914	7	0,009701	.....	.....	.....	.....
g	56930	7	0,00768	U	1048	13	0,000141
.	55048	7	0,00743	Ü	937	13	0,000126

Table 3: Frequencies, probabilities and codelengths of digrams

2 gram	Freq.	Codelen	P_i	2 gram	Freq.	Codelen	P_i
φφ	1294889	3	0,093166	en	52685	8	0,003791
φa	553340	5	0,039812	.....	.....	.....	.....
φe	433174	5	0,031167	ni	29578	9	0,002128
φi	407126	5	0,029292	el	29565	9	0,002127
φn	350248	5	0,0252	ay	28894	9	0,002079
.....	.....	.....	.....	ek	28592	9	0,002057
ar	92437	7	0,006651	.....	.....	.....	.....
la	85776	7	0,006172	oy	3637	12	0,000262
.....	.....	.....	.....	nk	3600	12	0,000259
er	73370	8	0,005279	ga	3554	12	0,000256
an	73126	8	0,005261	.....	.....	.....	.....
.....	.....	.....	.....	od	1038	14	7,47E-05
le	71235	8	0,005125	"t	1033	14	7,43E-05
in	69539	8	0,005003	zg	1029	14	7,4E-05
.....	.....	.....	.....	ht	1020	14	7,34E-05
da	52752	8	0,003795	.....	.....	.....	.....

Table 4: Frequencies, probabilities and codelengths of trigrams (φblank)

3 gram	Freq.	Codelen	P_i	3 gram	Freq.	Codelen	P_i
φφ	1294889	3	0,101757	.....	.....	.....	.....
φφφ	1227308	3	0,096446	iye	10133	10	0,000796
φφa	553340	5	0,043483	ele	10102	10	0,000794
φφe	433174	5	0,03404	ala	10034	10	0,000789
.....	.....	.....	.....	.....	.....	.....	.....
ler	32701	9	0,00257	miz	2526	12	0,000199
bir	27169	9	0,002135	una	2525	12	0,000198
eri	26025	9	0,002045	yaz	2521	12	0,000198
irφ	24892	9	0,001956	usu	2515	12	0,000198
.....	.....	.....	.....	.....	.....	.....	.....
ara	16946	10	0,001332	taφ	2363	12	0,000186
nda	16911	10	0,001329	rφv	2359	12	0,000185
φol	16345	10	0,001284	rar	2349	13	0,000185

reduced by 40%. (2). The success in the compression performance is increased by 1,5%. (3). During the decoding process, there will be no need to form a Huffman code tree for the second time. (4) Decoding process takes shorter time.

As the bit streams in the RSFD and SFD fields included in the header data are the Huffman coding tree itself, the decoder can get the symbol codes without forming the Huffman tree. In this method, 2(N+M)-4 bits would be enough for the transferring of root-stem "R" and suffix "S" if the root number is N and the suffix number is M. The direction data of the tree have been used to transfer the Huffman coding tree to the decoder. Direction data has a value of "0" and "1" and it can be expressed by one bit. In the practice, if the value is "0" while going to the left on the tree, we should go a level down and if the value is "1" on the right we should go a level up. The "0" and "1" values which give the direction data are stored in the RSFD/SFD fields according to their order of arrival and the values which are suitable for the leaves on the branches of the tree are stored in the RSAD/SFD field. In order to provide coherent working of the decoding and coding processes, all the direction data coding should be done from left to right. In this way by using the "left-node-right" notation, starting from the root of a tree we go to the lowest leaf on the left and after that we go to

Table 5: The comparison of the compression performance of monogram, digram and trigram

No		RawSize	1-gram	$\eta_{1\text{-gram}}$	2-gram	$\eta_{2\text{-gram}}$	3-gram	$\eta_{3\text{-gram}}$
1	article1	24.623	14.960	39,2	14.358	41,7	13.575	44,9
2	article2	36.596	21.964	40,0	20.855	43,0	19.471	46,8
3	article3	41.378	26.240	36,6	25.489	38,4	24.180	41,6
4	article4	48.880	29.746	39,1	28.580	41,5	26.792	45,2
5	text1	93.884	58.844	37,3	57.103	39,2	53.635	42,9
6	thesis	135.405	86.896	35,8	85.154	37,1	80.368	40,6
7	novel1	160.239	100.750	37,1	96.600	39,7	90.652	43,4
8	novel2	236.778	148.326	37,4	141.668	40,2	132.749	43,9
9	novel3	251.606	158.440	37,0	152.102	39,5	143.061	43,1
10	article5	666.120	367.508	44,8	347.380	47,9	319.505	52,0
11	article6	731.929	371.184	49,3	346.045	52,7	312.856	57,3
12	article7	1.055.244	638.116	39,5	522.762	50,5	486.395	53,9
13	article8	1.795.215	1.109.428	38,2	929.803	48,2	842.817	53,1
14	article9	2.147.095	1.389.754	35,3	1.143.691	46,7	1.071.935	50,1

Table 6: The comparison of the compression performance of 3-gram, WBDH with LZW and PKZip

No	RawSize	3-gram	$\eta_{3\text{-gram}}$	WBDH	η_{WBDH}	η_{LZW}	η_{PKZip}
1	24.623	13.575	44,9	12.359	49,8	49,5	59,6
2	36.596	19.471	46,8	17.850	51,2	49,2	64,7
3	41.378	24.180	41,6	20.899	49,5	44,4	59,3
4	48.880	26.792	45,2	23.341	52,2	47,1	66,8
5	93.884	53.635	42,9	45.074	52,0	45,7	58,6
6	135.405	80.368	40,6	61.213	54,8	51,0	69,4
7	160.239	90.652	43,4	75.373	53,0	49,4	61,6
8	236.778	132.749	43,9	110.571	53,3	47,6	57,5
9	251.606	143.061	43,1	118.845	52,8	47,5	58,4
10	666.120	319.505	52,0	272.494	59,1	57,2	68,6
11	731.929	312.856	57,3	270.805	63,0	60,4	72,3
12	1.055.244	486.395	53,9	420.227	60,2	58,6	69,6
13	1.795.215	842.817	53,1	716.377	60,1	55,9	68,5
14	2.147.095	1.071.935	50,1	910.814	57,6	53,9	65,5

a higher level and go to the right. Then we should control if the ending point of the process was a leaf or not, and the same process should be repeated again if the node point reached wasn't a leaf. If it is a leaf, we should go a level up till we can find an unprocessed branch on the right.

Comparison of Experimental Results (II. Step): When the files in the test-groups are compressed with the Word Based Dynamic Huffman method (WBDH), the average output has reached to a compression ratio of 59% (Table 6) Diri, 1999. The WBDH method has a 20% better result of success when it is compared to the compression performance of 1-gram method and it has a 8% better result of success when it is compared to the compression performance of 3-gram method.

When WBDH method is compared to the LZW (Lempel-Ziv) method, which uses dictionary Philips, 1992, it has a 4% higher compression performance. PKZIP (uses a variation of LZ 77 with static Huffman Coding method www.pkware.com has an 8% better result in compression performance than Word Based Dynamic Huffman method (Table 6). The compression performances of the four methods mentioned above are given in Fig.4 as graphics.

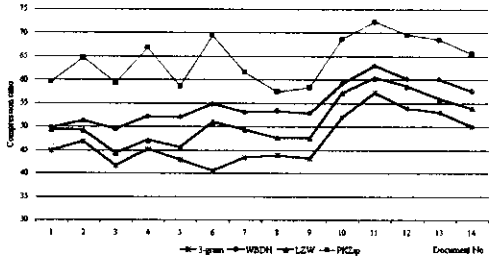

Fig. 4: The graphics of the compression performance received from the four different methods

Conclusion: We presented here four method types. All of them have been experimented using the Huffman coding tree and have been compared with each other. In the first phase of the study monograms, digrams and trigrams of Turkish language have been computed and by processing the collected data three different tuned templates have been prepared. The referential corpus files have been compressed by implementing the Huffman coding method on the prepared templates. In the second phase of study using the advantage of Turkish agglutinative word structure, the words have been placed in two different Huffman coding trees by classifying the root-stem and suffix occurrences. To perform the morphological classification adequately, two dictionaries have been built with 13.206 words of root and 580 suffixes symbols. Implementing the developed technique, Word Based Dynamic Huffman method the files in the referential corpus have been compressed and an average compression ratio of 59% has been measured. As a result this study gives an example of data compression on Turkish documents using the morphological structure of Turkish Language.

Future Work: Having special dictionaries depending on the subject of the source text will most probably improve the compression performance.

References

Diri, B., 1999. Türkçe'nin Biçimbilim Yapýsýna Dayalý Bir Metin Sýkýþtýrma Sistemi. Phd.Thesis, Dept, Computer Eng., YTU, Istanbul.

Gibson, J. D, T. Berger, *et al*, 1998. Digital Compression for Multimedia, Morgan Kauffmann.

http://www.pkware.com

Knut, D. E., 1985. Dynamic Huffman Coding, J. Algorithms, 6:163-180.

Phillips, D., 1992. LZW Data Compression, The Computer Application Journal Circuit Cellar Inc., 27:36-48.

Salomon, D., 1998. Data Compression, Springer, NY, USA.

Türk Dil Kurumu, 1992. Türkçe Sözlük, Milliyet Tesisleri. Türk Dil Kurumu, 1996. Ýmla Kýlavuzu, Türk Tarih Kurumu Basýmevi, Ankara.

Türk Dil Kurumu, 1996. İmla Kılavuzu, Türk Tarih Kurumu Basımevi, Ankara.